Amendments to the Specification:

Please replace the paragraph on page 1, lines 11 to 17, with the following amended paragraph:

The installation of program code in a computing system requires [[the]] an original program code to be loaded into the computing system from a CD or DVD, [[or]] downloaded onto the computing system over the internet to the computing system, or downloaded onto the computing system from a shared source in a corporate network ~~to the local computing system~~. Subsequent updates may again require access to the original program to install the update. Unfortunately, the original program code is not always available as the user may not have ~~have not~~ kept the original CD or not kept the download authorization for the original program code.

Please amend the paragraph beginning on page 1, line 22, and ending on page 2, line 4, with the following amended paragraph:

In embodiments of ~~accordance with~~ the present invention, a method is provided for retrieving a program code resource from a source of program code and placing the resource in a secure cache in the computer system. A user profile is impersonated by the computing system so that the computing system follows the user profile during retrieval of the resource. A source for the resource is picked, and the resource is opened. Whether the user has access to the resource is tested based on the user profile. If the user has access, the resource is read to the secure cache. The resource is written into the secure cache and verified that it is the same as the resource in the source. If the user does not have access, a new source for the resource is picked, and the resource in the new source is opened and read to the secure cache if the user has access to the resource in the new source.

Please amend the paragraph on page 2, lines 5-16, with the following amended paragraph:

In accordance with other aspects, [the]] embodiments of the present invention relate ~~relates~~ to a source engine system in a computing system for retrieving a program code resource from one of a plurality of sources of program code. An open resource module in the source engine opens a program code resource in a source if the user has access. A pick source module picks a second source for the resource program code if the user does not have access. A read

module transfers the program code resource to a writer module for a secure cache in the computing system. The writer module writes the program code resource into the secure cache. A verify module verifies the program code resource in the secure cache matches the program code resource in the source, and a delete module deletes the program code resource from the secure cache if the program code resource in the secure cache does not match the program code resource in the source. The source engine system profile impersonates the user profile during retrieval of the program code resource and returns to the system profile after the program code resource is written to the secure cache.

Please amend the paragraph on page 3, lines 2-4, with the following amended paragraph:

FIG. 1 illustrates one embodiment of the invention where the source engine in a local computing system retrieves a resource and places it in a secure cache for use by an installer module.

Please amend the paragraph beginning on page 3, line 19, and ending on page 4, line 3, with the following amended paragraph:

Fig. 1 illustrates one embodiment of the invention having a source engine in a local computer. The local computer **100** includes, in embodiments, the source engine software **102**, ~~as well as~~ a secure cache system **104** and an installer **106**. The source engine **102** has ~~exclusive~~ control over acquiring software resources and loading them into a secure cache **104**. The software resources may ~~would~~ be retrieved by the source engine from a CD or a DVD drive **108** and written into the secure cache **104**. Alternatively, the source engine **102** may ~~might~~ retrieve the program code resource as a download over the Internet **110** from a web site supplying the program code resource. Yet a third possible source of program code would be a corporate network where the corporation has a license to provide ~~share~~ the program code to local computers on the network. In one example, ~~this instance~~ the source engine retrieves the resource program code from the corporate network **112**, and ~~again~~ loads it into the secure cache **104.**

Please amend the paragraph on page 4, lines 4-8, with the following amended paragraph:

When the program code is to be installed in the local computer's system then the installer module **106** asks the source engine **102** the location of the program code and the secure cache

104. The installer then reads the program code from the secure cache **104**. The installer module may be a setup program[[.]] , ~~It may be~~ a repair program for installing patches, [[or]] an update routine to install updates, or ~~it may be~~ an alert system for installing repairs and updates.

Please replace the paragraph on page 4, lines 9-14, with the following amended paragraph:

~~It is significant that~~ In embodiments, the installer module **106** does not have access to the secure cache **104** except to read the program code placed there by the source engine **102**. The source engine **102** and the secure cache **104** may work together to provide a secure system for the original program code. ~~This~~ The secure system can prevent ~~prevents~~ inadvertent or deliberate alteration of the original program code retrieved by the source engine **102** and loaded into the secure cache **104**. Embodiments of ~~The~~ the operations performed by the source engine **102** to cache a resource in the secure cache **104** are shown in Figs. 3-7.

Please replace the paragraph on page 4, lines 21-24, with the following amended paragraph:

In addition to the memory **204**, the system may include at least one other form of computer[[-]]readable media. Computer readable media can be any available media that can be accessed by the system **200**. By way of example, and not limitation, computer[[-]]readable media might comprise computer storage media and communication media.

Please replace the paragraph beginning on page 5, line 25, and ending on page 6, line 9, with the following amended paragraph:

In the embodiments describing ~~description of~~ the source engine, which follow[[s]] in reference to FIGS. 3-[[_]]7, the logical operations of the various embodiments may be ~~of the present invention are~~ implemented (1) as a sequence of computer implemented acts or program modules running on a computing system and/or (2) as interconnected machine logic circuits or circuit modules within the computing system. The implementation is a matter of choice dependent on the performance requirements of the computing system implementing the invention. Accordingly, the logical operations making up the embodiments of the present invention described herein are referred to variously as operations, structural devices, acts, or

modules. It will be recognized by one skilled in the art that these operations, structural devices, acts and modules may be implemented in software, in firmware, in special purpose digital logic, or [[and]] any combination thereof without deviating from the spirit and scope of the present invention as recited within the claims attached hereto.

Please replace the paragraph on page 6, lines 10-15, with the following amended paragraph:

In Fig. 3, the operational flow for retrieving resource information is illustrated. The flow begins when request operation **302** receives a request from the calling program for resource information. In one embodiment, request ~~Request~~ operation requests the information from the source engine **102 (Fig. 1)**. The source engine **102 (Fig. 1),** in a download test operation **304**, detects whether the download exists. If the download containing the resource information exists, then the operation flow branches YES to the cache resource operation ~~module~~ **306**.

Please replace the paragraph on page 6, lines 16-19, with the following amended paragraph:

Cache resource operation **306** can detect ~~will see~~ that the resource is cached, can verify that the resource and can make the resource available. When the resource is available, the get operation **308** provides the resource to the calling program and the operation flow returns operation to ~~to the main flow of~~ the source engine **102 (Fig. 1)**.

Please replace the paragraph beginning on page 6, line 20, and ending on page 7, line 6, with the following amended paragraph:

If the download does not exist, then [[it]] the resource must be created in accordance with parameters from the calling program. In this example, ~~situation~~ the operation flow branches NO from the download test operation **304** to the create download package operation **310**. Create download package operation **310** receives the parameters of the download package from the calling program and builds the download package. With the download package defined and given and ID, the operation flow proceeds to create source operation **312**. Create source operation **312** receives parameter information defining the sources from the calling program and creates the sources IDs from which the resource information can be found. As discussed above,

there will typically be multiple sources for the same resource information. After create source operation **312**, create resource operation **314** receives a [[the]] list of resources defined by the calling program and generates resource IDs for the resource information so that the resources may be retrieved from a source. The completion of operations **310, 312,** and **314** completely define the download, which is given an ID, and which is then provided to the cache resource operation **306**.

Please replace the paragraph on page 7, lines 7-15, with the following amended paragraph:

The operational flow of the cache resource module **306 (Fig. 3)** is illustrated in Fig. 4. The operation flow begins at mark operation **402**, which marks the retrieval of the resource information as pending. Cache test operation **404** [[then]] tests whether the resource is in a cache or must be placed in a cache. If the resource is in the cache, then operation flow branches YES to verify resource operation **406**. Resource information is verified based on a check phrase written with the resource information in the cache. The [[This]] check phrase, in embodiments, is a hash number computed from the original resource information using a hash algorithm. A hash number for the resource information in the cache is calculated according to the same hash algorithm. If the calculation produces a hash number matching the check phrase, then the resource in the cache is verified.

Please replace the paragraph on page 7, lines 16-23, with the following amended paragraph:

Resource verify test **408** detects whether the verification operation **406** was successful. If the resource was verified, then the operation flow branches YES to mark operation **412** to mark the resource available from the cache. The program flow then returns to get the resource information operation **308** in Fig. 3. If the resource information cache was ~~did~~ not verified, then the operation flow branches NO to delete operation **410**. Delete operation **410** erases the written resource information in the cache, and the operation flow proceeds to operations to rewrite or to load the resource information in the cache. Operations loading the resource information in the cache are described hereinafter.

Please replace the paragraph beginning on page 7, line 24, and ending on page 8, line 14, with the following amended paragraph:

Returning to the in cache test operation **404**, if the resource information is detected as not in the cache, then the operation flow branches NO to the load cache operations. The load cache operations amount to two parallel operations: (1) reading resource information from a source and (2) writing the resource information into a cache. The writing operation is performed by writer module **412**, which is described hereinafter with reference to Fig. 6. The retrieving of the resource information from a source can be accomplished in either of two ways depending on whether the resource is stored at a first level in a source or whether it is stored at a second level within a source. [[By]] In embodiments, a second level [[it]] is ~~meant that~~ the resource [[is]] inside a container stored in the source. The container is often referred to as a cabinet file and the resource as a file inside the cabinet file. If the resource is in a resource file in a source, then [it]] the resource is retrieved by a reader module **414**, which is described hereinafter with reference to Fig. 5. If the resource file is in a cabinet file or a folder in the source, then the resource is retrieved by an extractor module **416**. The [[E]]extractor module **416** is described hereinafter with reference to Fig. 7. ~~The detection of~~ [[w]]Whether the resource file is in a source or in a cabinet file in a source is detected by contained test operation **418**. Contain test operation **418** branches the operation flow to the extractor module **416** is the resource file is contained within a cabinet file in the source. If the resource file is not contained in another file, then the operation flow branches NO from contain test operation **418** to the reader module **414**.

Please replace the paragraph on page 8, lines 15-21, with the following amended paragraph:

In Fig. 5 the operational flow of the reader module begins with [the]] an initialize operation **502**. The [[I]]initialize operation **502** loads all the necessary parameters for the reader to retrieve the resource from a source. After initialization, impersonate operation **504** installs all of the personal parameters for the user into the retrieval process. In other words, the retrieval process is operating based on the profile of the user rather than the profile of the computing system that the user is using. [This]] The impersonation is important as it provides additional security, i.e., only a user with the user's profile may perform the cache loading and the retrieval of the resource from the cache.

Please replace the paragraph beginning on page 8, line 22, and ending on page 9, line 11, with the following amended paragraph:

After the reader is initialized, and the user's profile is loaded, then a source identity ~~identified~~ test **506** detects whether the user has specified a source from which to retrieve the resource. If the user specified a source for the resource, then the operation flow branches YES from the source identify ~~identified~~ test **506** to open resource operation **508**. The [[O]]open resource operation **508** gets the resource file from the source and opens ~~it if it is found and~~ the resource file, if found and if the user has access. Resource open test operation **510** is checking to see that the resource was found and could be opened. Access test operation **512** ~~is checking~~ checks whether the user has access to the resource, i.e., is identified as a user that can have the resource. If the resource open test and the access test[[s]] are passed successfully, then the operation flow branches YES from access test operation **512** to a read date operation **514**. Read data operation **514** begins the process of transferring the data to the writer. As each block of data is read, put data operation **516** transfers the data into the writer, and more data test operation **518** tests whether more data is in the resource file that is being written to the cache. If there is more data, the ~~operation~~ flow loops back to read data operation **514** and this loop **514, 516, 518** continues until all the data of the resource file is read to the writer. As will be discussed hereinafter, while the [[this]] data is being read to the writer the writer is writing the data and verifying [[it]] the data in the cache. After all of the data from the resource file is read, then the operation flow returns to the flow in Fig. 3.

Please replace the paragraph on page 9, lines 12-20, with the following amended paragraph:

The above operation flow is the simplest and most direct flow through the reader module illustrated in Fig. 5. However, if the resource was not found, [[or]] would not open, or the user did not have access to the resource in the source specified by the user, then the operation flow branches NO from access test operation **512** or NO from resource open test operation **510** ~~and proceeds~~ to more sources test operation **520**. If there are more sources identified by the download package for the resource, then pick operation **522** picks another source for the resource. The sources will be prioritized in the download package and the pick operation **522**

will accordingly pick the next possible source on the priority list. The operation flow then proceeds to open resource operation **508** and attempts to open the resource in this new source.

Please replace the paragraph on page 9, lines 21-25, with the following amended paragraph:

If all the sources have been tried and either the resource was not found, [[or]] could not be opened, or the user had no access, then more sources test **520** will fail[[.]] and [[T]]the operation flow will branch NO from the more sources test **520** to the send operation **524**, which will send an error message. The error message will indicate the cause of the problem[[.]], e.g., [[T]]the resource was not found, the resource would not open, or the user does not have access.

Please replace the paragraph on page 10, lines 1-20, with the following amended paragraph:

While the reading of data from the resource is occurring, the writer module is operating in parallel to load the data into a cache. Fig. 6 shows the operational flow of the writer module. The write operation begins with initialize operation **602**. Initialize operation **602** initializes the writer according to the parameters received with the download package. After initialization the writer at wait operation 604 waits for data from the reader or extractor. When data is received more data test operation **606** detects whether there is more data in the resource information than what has been received at present. If there is more data, then operation flow branches YES to the write data operation **608** that begins to write the data into a cache. Thus, while the reader or extractor is reading information, the writer module is writing the data into the cache ~~as soon as it~~ when the data is received. This loop of operations **604, 606,** and **608** continues until the more data test operation **606** detects that all the data in the resource has been received. When ~~this occurs,~~ all the data has been received, the operation flow branches NO to the mark operation **610**. Mark operation **610** indicates to the user that the caching phase of the operation has been successful. After the resource information is in the cache then a verify operation **612** verifies the resource information. Verify operation **612** operates in ~~the same~~ a similar manner ~~as described above for~~ to the verify operation in Fig. 4. In other words, if [[the]] a hash count for the resource information matches [[the]] a hash count downloaded when writing the resource information into the cache, [[and]] the resource is verified. ~~Verified~~ Verify test operation **614** tests whether the

resource information was successfully verified. If it was not, then the operation flow branches NO to delete operation **616**. Delete operation **616** deletes the resource information just written into cache and the operation flow returns to the source engine **102** (Fig. 1).

Please replace the paragraph on page 10, lines 21-25 with the following amended paragraph:

If the verify resource operation is successful, the operation flow branches YES from ~~verified~~ verify test **614** to mark operation **618**. Mark operation **618** marks the resource information as available from the cache. An un-impersonate [[O]]operation **620** then un-impersonates the user so that the source engine **102** (Fig. 1) will now begin to operate under its computing system profile rather than the user's profile. Operation flow then returns to the get resource information operation **308** in Fig. 3.

Please replace the paragraph on page 11, lines 1-10, with the following amended paragraph:

If the resource information had been retrieved using the extractor module, the same writing module operation flow occurs. ~~However~~ In contrast, the reading of the resource information proceeds as indicated in Fig. 7. Fig. 7 shows the operational flow for the extractor module **416** (Fig. 4). ~~In the extractor module~~[[t]]The operation flow begins at initialize operation **702** that initializes the extractor according to the parameters received with the download package. Open operation **704** then opens the container in the source that contains the resource file. If the opening of the container is not successful as detected by test operation **706**, the operation flow will branch NO and sends [[and]] an error message operation **708** to the user. An alternative source [[will]] may then be tried if possible. If the container is opened successfully, then the operation flow branches YES from test operation **706** to impersonate operation **710**.

Please replace the paragraph on page 11, lines 11-14, with the following amended paragraph:

Impersonate operation **710** impersonates the user in ~~the same~~ a similar manner as impersonate ~~operate~~ operation **504** does in the reader module as shown [[on]] in conjunction with Fig. 5. ~~Basically~~ In embodiments, the impersonate operation **710** applies the profile of the user

to the source engine so that the opening of the resource and the security check will be performed in accordance with the user's profile.

Please replace the paragraph on page 11, lines 15-22, with the following amended paragraph:

Open resource and security check module **712** operates in ~~the same~~ a similar manner as with operations discussed above for opening the resource and checking user access [[as]] described in Fig. 5. If the resource is found, if it opens successfully, and if the user has access, then operation flow will proceed to the read out module **714**. Read out module **714** operates in ~~the same~~ a similar manner [[as]] to operations **514, 516**, and **518**. The extractor module operates in parallel with the writer module so that data is retrieved from a source and read into the cache in parallel operations. The extractor module is similar to the reader module with the additional operations necessary to open a container that contains the resource file.